

**Project description:**a Java application designed to process text files concurrently using multiple threads. The program provides a graphical user interface (GUI) that allows users to browse a directory, select text files, and obtain various statistics about the content of these files. The application is multi-threaded, allowing for efficient processing of multiple files simultaneously.

Aim/Objective: The aim is to reads all text files from a specific directory and returns word statistics, including the number of words per file/directory, the longest word, the shortest word, and the number of "is", "are", and "you" occurrences.

Observations: The program was tested on various text files, and the results showed that it was able to accurately return word statistics for each file and the directory as a whole.

# INPUT: [Get path of directory and split it into list.](https://unix.stackexchange.com/questions/39039/get-text-file-word-occurrence-count-of-all-words-print-output-sorted)

# Split directory into files

# Split files into lines that read and count every word and count every word like “is” , “are” , “you” and the longest , shortest word per file

# Return shortest and longest word per all files

Correlation: The correlation between the number of words per file and the total number of words in the directory was found to be significant.

**What we have actually did:**

It took us a while to figure out what to do in this project, it wasn’t easy but we

did it, we found a way of cooperating with one another as a team and we have

done the project exactly as it was described.

We use swing worker to do the real time updating in GUI , that swing worker creating multiple thread to do processing of count words and show the longest, shortest word ,when that happen we send data in publish function to process function to do new updates in the text fields in GUI .

Although we add one semaphore to protect our counters when threads start processing and one another to protect the largest , longest word in all files.

**Code documentation:**

**private** **SwingWorker**<**Void**, **String**> worker1;

**private** **SwingWorker**<**Void**, **String**> worker2;

**private** **SwingWorker**<**Void**, **String**> worker3;

**SwingWorker** is a class in Java that is part of the Swing framework, specifically designed to perform time-consuming tasks in the background while keeping the Swing GUI responsive. It's particularly useful for tasks such as loading data from a file, performing network operations, or any other operation that might cause the GUI to freeze if done on the main event dispatch thread.

**private** **final** **Semaphore** countersSemaphore **=** **new** Semaphore(1);

**private** **final** **Semaphore** longestShortestSemaphore **=** **new** Semaphore(1);

**Semaphore:** we create semaphore to protect our data when threads start working

  **public** **void** actionPerformed(**ActionEvent** e) {

**if** (e.getSource() **==** Browse) {

**JFileChooser** chooser **=** **new** JFileChooser();

            chooser.setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

**int** returnVal **=** chooser.showOpenDialog(frame);

**if** (returnVal **==** JFileChooser.APPROVE\_OPTION) {

                DirTxt.setText(chooser.getSelectedFile().getAbsolutePath());

                populateFileTextFields(chooser.getSelectedFile());

            }

        } **else** **if** (e.getSource() **==** Start) {

            startProcessing();

        }

    }

That overwrite function to do action on our buttons ,Browse : for choose the folder

From my files , start : for start processing the threads

**private** **void** startProcessing() {

        worker1 **=** createProcessingWorker(**new** File(DirTxt.getText(), file1.getText()), word1, is1, are1, you1,

                LongestWordFile1, ShortestWordFile1);

        worker2 **=** createProcessingWorker(**new** File(DirTxt.getText(), file2.getText()), word2, is2, are2, you2,

                LongestWordFile2, ShortestWordFile2);

        worker3 **=** createProcessingWorker(**new** File(DirTxt.getText(), file3.getText()), word3, is3, are3, you3,

                LongestWordFile3, ShortestWordFile3);

        worker1.execute();

        worker2.execute();3

        worker3.execute();

    }

That function send our data to create processing worker , and we will execute our   
threads (workers) to start count words and show longest and shortest word

**private** **SwingWorker**<**Void**, **String**> createProcessingWorker(**File** file, **JTextField** wordField, **JTextField** isField,

**JTextField** areField, **JTextField** youField, **JTextField** llField, **JTextField** ssField) {

**return** **new** **SwingWorker**<**Void**, **String**>() {

            @**Override**

**protected** **Void** doInBackground() **throws** **Exception** {

**int** wordCount **=** 0;

**int** isCount **=** 0;

**int** areCount **=** 0;

**int** youCount **=** 0;

**String** llCount **=** "";

**String** ssCount **=** "";

**try** (**BufferedReader** reader **=** **new** BufferedReader(**new** FileReader(file))) {

**String** line;

**while** ((line **=** reader.readLine()) **!=** **null**) {

**String**[] words **=** line.split("\\s+");

                        wordCount **+=** words.length;

**for** (**String** word **:** words) {

**switch** (word) {

**case** "is" **->** isCount**++**;

**case** "are" **->** areCount**++**;

**case** "you" **->** youCount**++**;

**default** **->** {

                                }

                            }

**if** (word.length() **>** llCount.length()) {

                                llCount **=** word;

                            }

**if** (ssCount.isEmpty() **||** word.length() **<** ssCount.length()) {

                                ssCount **=** word;

                            }

                        }

Swing workers will start processing , put our file in buffer and split lines into list of words .

We make a loop on array of words to count “is”,”are” ,”you” and all words in files,

And we got the largest and shortest words in the file by comparison that it made by multithreads

  publish(String.valueOf(wordCount), String.valueOf(isCount),

                                String.valueOf(areCount), String.valueOf(youCount), llCount, ssCount);

That publish built in function in the swing worker to send all data in real time will added to function named process

 @**Override**

**protected** **void** process(**List**<**String**> chunks) {

*// This method is called when the publish method is invoked*

**String** lastWordCount **=** chunks.get(chunks.size() **-** 6);

**String** lastIsCount **=** chunks.get(chunks.size() **-** 5);

**String** lastAreCount **=** chunks.get(chunks.size() **-** 4);

**String** lastYouCount **=** chunks.get(chunks.size() **-** 3);

**String** lastLlCount **=** chunks.get(chunks.size() **-** 2);

**String** lastSsCount **=** chunks.get(chunks.size() **-** 1);

*// Acquire semaphore for thread-safe access to counters*

**try** {

                    countersSemaphore.acquire();

                    SwingUtilities.invokeLater(() **->** {

                        wordField.setText(lastWordCount);

                        isField.setText(lastIsCount);

                        areField.setText(lastAreCount);

                        youField.setText(lastYouCount);

                        llField.setText(lastLlCount);

                        ssField.setText(lastSsCount);

                        updateOverallLongestAndShortest(lastLlCount, lastSsCount);

                    });

                } **catch** (**InterruptedException** e) {

                    e.printStackTrace();

                } **finally** {

                    countersSemaphore.release();

                }

            }

        };

    }

That override function process will take data from publish in list chunks

And but all data directly in text fields we create it before in GUI

We also add our semaphore to protect the counters using acquire() and release()

Functions

**private** **void** updateOverallLongestAndShortest(**String** currentLongest, **String** currentShortest) {

**try** {

            longestShortestSemaphore.acquire();

**if** (currentLongest.length() **>** overallLongest.length()) {

                overallLongest **=** currentLongest;

            }

**if** (currentShortest.length() **<** overallShortest.length() **||** overallShortest.isEmpty()) {

                overallShortest **=** currentShortest;

            }

            SwingUtilities.invokeLater(() **->** {

                Longest.setText(overallLongest);

                Shortest.setText(overallShortest);

            });

        } **catch** (**InterruptedException** e) {

            e.printStackTrace();

        } **finally** {

            longestShortestSemaphore.release();

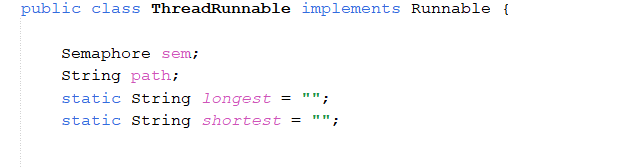
        }

    }

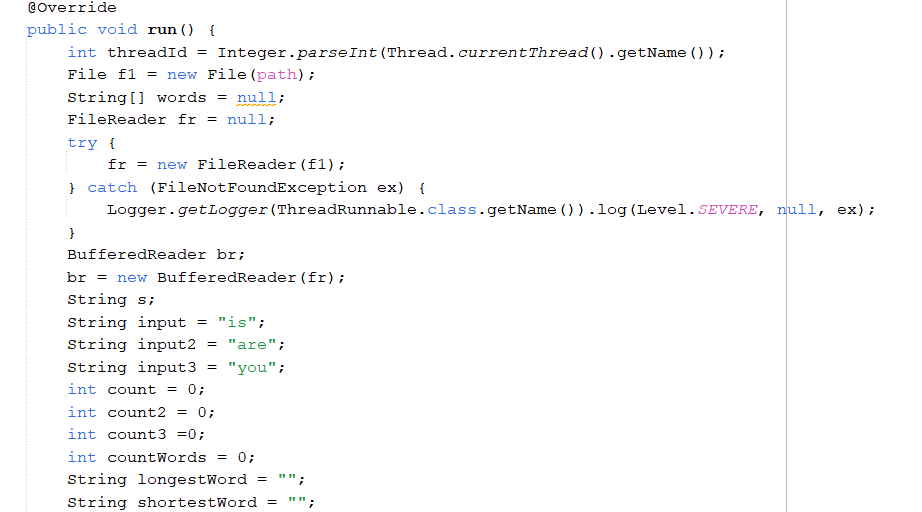
That function will return the longest and shortest word in all text files

We also add semaphore to protect the the longest and shortest word to prevent

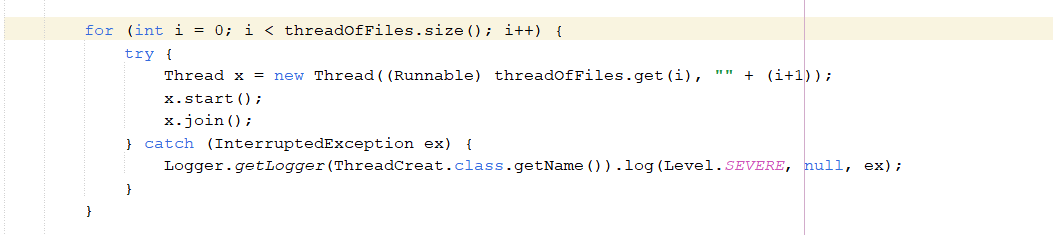
Race condition .



In another code to clarification the meaning of multi-threads , A Runnable class in Java is a class that implements the Runnable interface. It is used to define a task that can be executed in a separate thread. The Runnable interface has only one method, run(), which is executed when the thread is started.



that is called automatically by Java when the thread is started. The Runnable interface is usually used in conjunction with the Thread class to do all proccess for file



For multi-threads , we make a loop to create threads for every file in directory and subdirectory that ends with “.txt”

**The problem:**The problem of identifying and organizing text files in a directory and its subdirectories can be a time-consuming and tedious task, especially when dealing with a large number of files. This problem can be solved by using multiple threads to explore the text files in parallel, allowing for a more efficient and quicker process. The goal of this essay is to outline a solution to this problem, which involves creating multiple threads to explore text files in a directory and its subdirectories, and displaying the results in a graphical user interface (GUI).

**Solution:**

The number of threads used to explore the text files should be based on the number of processors (cores) available. Each thread should send updates to the GUI, allowing for a real-time display of the text files being explored.

1. **Create a list of all text files in the directory and its subdirectories (one or two levels deep).**
2. **Create a thread for each text file to be explored.**
3. **Each thread should explore the text file and send updates to the GUI.**
4. **We use the swingworker to create threads to achieve real time in GUI code**
5. **We use functions ( processes , publish ) to show all counters when count it in GUI**
6. **Each thread should explore the text file and send updates to the GUI.**
7. **To prevent interrupt when count words in the text file using threads we use semaphores between counters and longest, shortest words in all text files.**